

Problem 2303: Calculate Amount Paid in Taxes

Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

2D integer array

brackets

where

`brackets[i] = [upper`

`i`

, percent

`i`

`]`

means that the

`i`

th

tax bracket has an upper bound of

upper

i

and is taxed at a rate of

percent

i

. The brackets are

sorted

by upper bound (i.e.

upper

i-1

< upper

i

for

$0 < i < \text{brackets.length}$

).

Tax is calculated as follows:

The first

upper

0

dollars earned are taxed at a rate of

percent

0

.

The next

upper

1

- upper

0

dollars earned are taxed at a rate of

percent

1

.

The next

upper

2

- upper

1

dollars earned are taxed at a rate of

percent

2

.

And so on.

You are given an integer

income

representing the amount of money you earned. Return

the amount of money that you have to pay in taxes.

Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

brackets = [[3,50],[7,10],[12,25]], income = 10

Output:

2.65000

Explanation:

Based on your income, you have 3 dollars in the 1

st

tax bracket, 4 dollars in the 2

nd

tax bracket, and 3 dollars in the 3

rd

tax bracket. The tax rate for the three tax brackets is 50%, 10%, and 25%, respectively. In total, you pay $\$3 * 50\% + \$4 * 10\% + \$3 * 25\% = \2.65 in taxes.

Example 2:

Input:

brackets = $[[1,0],[4,25],[5,50]]$, income = 2

Output:

0.25000

Explanation:

Based on your income, you have 1 dollar in the 1

st

tax bracket and 1 dollar in the 2

nd

tax bracket. The tax rate for the two tax brackets is 0% and 25%, respectively. In total, you pay $\$1 * 0\% + \$1 * 25\% = \$0.25$ in taxes.

Example 3:

Input:

brackets = $[[2,50]]$, income = 0

Output:

0.00000

Explanation:

You have no income to tax, so you have to pay a total of \$0 in taxes.

Constraints:

$1 \leq \text{brackets.length} \leq 100$

$1 \leq \text{upper}$

i

≤ 1000

$0 \leq \text{percent}$

i

≤ 100

$0 \leq \text{income} \leq 1000$

upper

i

is sorted in ascending order.

All the values of

upper

i

are

unique

.

The upper bound of the last tax bracket is greater than or equal to

income

.

Code Snippets

C++:

```
class Solution {
public:
    double calculateTax(vector<vector<int>>& brackets, int income) {

    }
};
```

Java:

```
class Solution {
    public double calculateTax(int[][] brackets, int income) {

    }
}
```

Python3:

```
class Solution:
    def calculateTax(self, brackets: List[List[int]], income: int) -> float:
```

Python:

```
class Solution(object):
    def calculateTax(self, brackets, income):
```

```

"""
:type brackets: List[List[int]]
:type income: int
:rtype: float
"""

```

JavaScript:

```

/**
 * @param {number[][]} brackets
 * @param {number} income
 * @return {number}
 */
var calculateTax = function(brackets, income) {

};

```

TypeScript:

```

function calculateTax(brackets: number[][], income: number): number {

};

```

C#:

```

public class Solution {
    public double CalculateTax(int[][] brackets, int income) {

    }
}

```

C:

```

double calculateTax(int** brackets, int bracketsSize, int* bracketsColSize,
int income) {

}

```

Go:

```

func calculateTax(brackets [][]int, income int) float64 {

```

```
}
```

Kotlin:

```
class Solution {  
    fun calculateTax(brackets: Array<IntArray>, income: Int): Double {  
  
    }  
}
```

Swift:

```
class Solution {  
    func calculateTax(_ brackets: [[Int]], _ income: Int) -> Double {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn calculate_tax(brackets: Vec<Vec<i32>>, income: i32) -> f64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[][]} brackets  
# @param {Integer} income  
# @return {Float}  
def calculate_tax(brackets, income)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $brackets  
     * @param Integer $income
```

```

* @return Float
*/
function calculateTax($brackets, $income) {

}
}

```

Dart:

```

class Solution {
  double calculateTax(List<List<int>> brackets, int income) {

  }
}

```

Scala:

```

object Solution {
  def calculateTax(brackets: Array[Array[Int]], income: Int): Double = {

  }
}

```

Elixir:

```

defmodule Solution do
  @spec calculate_tax(brackets :: [[integer]], income :: integer) :: float
  def calculate_tax(brackets, income) do

  end
end

```

Erlang:

```

-spec calculate_tax(Brackets :: [[integer()]], Income :: integer()) ->
float().
calculate_tax(Brackets, Income) ->
.

```

Racket:

```
(define/contract (calculate-tax brackets income)
  (-> (listof (listof exact-integer?)) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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:
    double calculateTax(vector<vector<int>>& brackets, int income) {

    }
};
```

Java Solution:

```
/**
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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 double calculateTax(int[][] brackets, int income) {

    }
}
```

```
}
```

Python3 Solution:

```
"""
Problem: Calculate Amount Paid in Taxes
Difficulty: Easy
Tags: array, sort

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 calculateTax(self, brackets: List[List[int]], income: int) -> float:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):
    def calculateTax(self, brackets, income):
        """
        :type brackets: List[List[int]]
        :type income: int
        :rtype: float
        """
```

JavaScript Solution:

```
/**
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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[][]} brackets
 * @param {number} income
 * @return {number}
 */
var calculateTax = function(brackets, income) {

};

```

TypeScript Solution:

```

/**
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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 calculateTax(brackets: number[][], income: number): number {

};

```

C# Solution:

```

/*
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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 double CalculateTax(int[][] brackets, int income) {

    }
}

```

```
}
```

C Solution:

```
/*
 * Problem: Calculate Amount Paid in Taxes
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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
 */

double calculateTax(int** brackets, int bracketsSize, int* bracketsColSize,
int income) {

}
```

Go Solution:

```
// Problem: Calculate Amount Paid in Taxes
// Difficulty: Easy
// Tags: array, sort
//
// 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 calculateTax(brackets [][]int, income int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun calculateTax(brackets: Array<IntArray>, income: Int): Double {

}

}
```

Swift Solution:

```
class Solution {  
    func calculateTax(_ brackets: [[Int]], _ income: Int) -> Double {  
  
    }  
}
```

Rust Solution:

```
// Problem: Calculate Amount Paid in Taxes  
// Difficulty: Easy  
// Tags: array, sort  
//  
// 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 calculate_tax(brackets: Vec<Vec<i32>>, income: i32) -> f64 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[][]} brackets  
# @param {Integer} income  
# @return {Float}  
def calculate_tax(brackets, income)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[][] $brackets  
     * @param Integer $income  
     * @return Float  
     */  
}
```

```
function calculateTax($brackets, $income) {

}

}
```

Dart Solution:

```
class Solution {
  double calculateTax(List<List<int>> brackets, int income) {

  }
}
```

Scala Solution:

```
object Solution {
  def calculateTax(brackets: Array[Array[Int]], income: Int): Double = {

  }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec calculate_tax(brackets :: [[integer]], income :: integer) :: float
  def calculate_tax(brackets, income) do

  end
end
```

Erlang Solution:

```
-spec calculate_tax(Brackets :: [[integer()]], Income :: integer()) ->
float().
calculate_tax(Brackets, Income) ->
.
```

Racket Solution:

```
(define/contract (calculate-tax brackets income)
  (-> (listof (listof exact-integer?)) exact-integer? flonum?))
```

