2303. Calculate Amount Paid in Taxes

```
Simulation
Easy
```

Problem Description

The given LeetCode problem involves calculating the amount of taxes that need to be paid based on a set of tax brackets. Each bracket is defined by two parameters: upper, which is the upper bound of taxable income for that bracket, and percent, which is the tax rate for that bracket. The brackets array contains these pairs in ascending order of upper.

To calculate the total tax:

- - For the first bracket, tax is applied on the entire income if the income is less than or equal to upper [0], otherwise, tax is applied only up to upper[0]. • For each subsequent bracket, tax is applied on the difference between the current bracket's upper and the previous bracket's upper, but only if the income is more than the previous upper. This continues until all the income is taxed or until all the brackets are exhausted.
 - The goal is to find out the total tax owed based on the income provided, where income is the total amount of money earned by an individual.
- ntuition

The intuition behind the solution is to iterate through the tax brackets and calculate the taxes for each bracket incrementally.

Since the income can span multiple brackets, the approach must consider partial amounts taxed at different brackets.

To implement this, we keep track of the previous bracket's upper bound as we iterate through the brackets so that we can

calculate the taxable amount within the current bracket. The key steps in the solution are as follows:

1. Initialize ans as the accumulator for the total tax and prev to keep the upper bound of the previous bracket, starting with 0 for the initial

condition. 2. Iterate over each bracket in the brackets list, which has upper and percent values. 3. For each bracket, calculate the amount of income that falls within this bracket. This is the minimum of the actual income and the current

- bracket's upper minus prev, which signifies the taxed amount in the previous brackets. 4. Calculate the tax for this bracket by multiplying the bracket's taxable income by the percent.
- 5. Update prev to be the current bracket's upper so that it can be used in the next iteration.
- 6. The tax rates are provided in percentages, so divide the final answer by 100 to get the actual tax amount.
- 7. Continue this process until all the brackets are covered or until the entire income is taxed.
- The algorithm for calculating taxes based on tax brackets is straightforward. The solution takes advantage of the sorted order of the tax brackets, following these steps:

the previous bracket, starting with 0 since there's no previous bracket at the beginning.

Solution Approach

for that bracket. In each iteration, calculate the taxable amount for the current bracket. This is done by subtracting prev from the minimum of

Initialize two variables: ans to keep track of the total taxes paid so far, which starts at 0, and prev to track the upper bound of

Loop through each bracket in the brackets array, extracting upper, the upper bound for the bracket, and percent, the tax rate

- income and the current upper. The subtraction of prev ensures only the income falling within the current bracket is considered. This is represented by the following expression: taxable_amount = max(0, min(income, upper) - prev)
- Calculate the tax for the current bracket by multiplying the taxable_amount by the percent rate. Since percent represents a
- percentage, you'll need to multiply the tax amount by the rate and then divide by 100 to convert it into the actual tax value. The tax for the current bracket is added to the ans accumulator:

Once the tax for this bracket is calculated, update prev to the current bracket's upper, which will be used in the next iteration

After completing the loop over all brackets, the final tax value stored in ans is divided by 100 to adjust for the percentage

[(10000, 10), (20000, 20), (30000, 30)]

The tax for the first bracket is 10000 * 10% = 1000.

The tax for the third bracket is 5000 * 30% = 1500.

Initialize the total tax amount to 0

total_tax += taxable_income * tax_rate

previous_upper_bound = upper_bound

to calculate the next bracket's taxable income.

ans += taxable_amount * percent

the current bracket's upper. It also handles cases where the income is exactly on the upper of the previous bracket, thus not bleeding into the current tax bracket's range.

The use of max(0, min(income, upper) - prev) ensures that the solution also handles cases where the income does not reach

No additional data structures are needed, as the algorithm only requires simple variable assignments and arithmetic operations.

The pattern followed is iterative, straightforward, and efficient, as it only requires one pass through the brackets array, making the complexity of this algorithm proportional to the number of brackets (O(n) time complexity, where n is the number of

brackets).

prev = upper

calculation:

return ans / 100

This solution effectively calculates the total taxes for a given income according to the provided set of tax brackets. **Example Walkthrough**

Let's walk through a small example to illustrate the solution approach. Suppose we have the following set of brackets:

This example states that: Tax is 10% on income up to \$10,000, Tax is 20% on income between 10, 000and20,000, • Tax is 30% on income between 20,000 and 30,000.

2. Start looping through the brackets.

1. Initialize ans to 0 and prev to 0.

First bracket: (10000, 10) • The taxable income for the first bracket is min(25000, 10000) - 0 = 10000.

Update ans by 1000 (now ans = 1000) and prev by the upper of the current bracket (now prev = 10000).

Update ans by 2000 (now ans = 3000) and prev by the upper of the current bracket (now prev = 20000).

Since the income is 25,000, which is less than the 'upper' of 30,000, the calculation here is on the remaining income. The

Update ans by 1500 (now ans = 4500). There's no need to update prev since we've already covered all the income.

So the total tax on an income of 25,000with the given tax brackets is 45. The algorithm correctly breaks down the income into

3. After all the brackets are processed, divide ans by 100 to adjust the percentage calculation. Therefore, ans/100 = 4500/100 = 45.

Let's calculate the total tax for an income of \$25,000 using the algorithm outlined in the solution approach:

The taxable income for the second bracket is min(25000, 20000) - 10000 = 10000. The tax for the second bracket is 10000 * 20% = 2000.

Second bracket: (20000, 20)

Third bracket: (30000, 30)

Solution Implementation

total_tax = 0

Python

class Solution:

taxable income for the third bracket is min(25000, 30000) - 20000 = 5000.

portions that fall within each tax bracket, calculates the tax on each portion accordingly, and sums up the taxed amounts to determine the total tax liability.

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

and that income is not taxed twice for the lower brackets.

// `taxAmount` will store the calculated tax based on brackets.

int currentBracketUpperLimit = bracket[0];

taxable_income = max(0, min(income, upper_bound) - previous_upper_bound)

// `previousBracketUpperLimit` holds the upper limit of the previous tax bracket.

// Calculate the taxed income at this bracket by taking the lesser of

// Update the total taxAmount with the tax from this bracket's range.

// Tax rates are given in percentage, divide by 100 to get the actual tax amount

the upper limit and the tax rate (as a percentage) for that bracket.

taxAmount += taxedIncomeAtCurrentBracket * taxRatePercent;

Calculate the tax for the current bracket and accumulate it into 'total_tax'

Update 'previous_upper_bound' to the current bracket's upper bound for the next iteration

Divide the 'total_tax' by 100 to convert the tax rate to a percentage and return the result

// Each bracket contains an upper limit and the tax rate percent for the bracket range.

int taxedIncomeAtCurrentBracket = Math.max(0, Math.min(income, currentBracketUpperLimit)

// income or the bracket's upper limit minus the previous bracket's upper limit.

// This is the amount of income that falls within the current bracket's range.

previous_upper_bound = 0 # Loop through the collected tax brackets for upper_bound, tax_rate in brackets: # Calculate the taxable income for the current bracket. # 'max(0, min(income, upper_bound) - previous_upper_bound)' ensures that the income does not exceed the current brack

Initialize 'previous_upper_bound' which will hold the upper bound of the previous bracket

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

int taxAmount = 0;

int previousBracketUpperLimit = 0;

for (int[] bracket : brackets) {

int taxRatePercent = bracket[1];

return total_tax / 100

```
// Update the previousBracketUpperLimit for the next iteration.
            previousBracketUpperLimit = currentBracketUpperLimit;
       // Convert the taxAmount to dollars and cents (as the percent was in whole number).
       return taxAmount / 100.0;
C++
class Solution {
public:
   // Function to calculate the tax based on given brackets and income
   double calculateTax(vector<vector<int>>& brackets, int income) {
        int tax = 0; // Variable to store the total tax
        int prevBracketUpperLimit = 0; // Variable to keep track of the previous bracket upper limit
       // Iterate over the tax brackets
        for (const auto& bracket : brackets) {
            int currentBracketUpperLimit = bracket[0]; // Upper limit for the current tax bracket
            int taxRate = bracket[1]; // Tax rate for the current bracket
           // Calculate the tax for the income falling within the current tax bracket range
           // max(0, ...) ensures we don't get negative values in case income is less than the previous bracket
           // min(income, currentBracketUpperLimit) ensures we don't calculate tax for income beyond the current bracket
            // Subtracting prevBracketUpperLimit gives us the taxable amount in the current bracket
            tax += max(0, min(income, currentBracketUpperLimit) - prevBracketUpperLimit) * taxRate;
            prevBracketUpperLimit = currentBracketUpperLimit; // Update the previous bracket upper limit for the next iteration
           // If income is less than the current bracket upper limit, no need to continue
            if (income < currentBracketUpperLimit) {</pre>
```

- previousBracketUpperLimit);

```
// Loop over each bracket
for (const [currentBracketUpper, taxRatePercent] of brackets) {
   // Calculate the taxable income for the current bracket
    const taxableIncome = Math.max(0, Math.min(income, currentBracketUpper) - previousBracketUpper);
```

totalTax += taxableIncome * taxRatePercent;

previousBracketUpper = currentBracketUpper;

total_tax += taxable_income * tax_rate

ans += max(0, min(income, upper) - prev) * percent

previous_upper_bound = upper_bound

* @param income - The total income to calculate tax for.

* @returns The total tax calculated based on the brackets.

let totalTax = 0; // Stores the cumulative tax amount

* Calculates the total tax based on tax brackets.

* Each tax bracket specifies the upper limit and the tax rate.

* Income is taxed in a progressive manner according to these brackets.

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

* @param brackets - An array of arrays where each inner array contains 2 numbers:

let previousBracketUpper = 0; // The upper limit of the previous bracket

// Update the previous bracket upper limit for the next iteration

return totalTax / 100; // Convert percentage to a decimal representation

and that income is not taxed twice for the lower brackets.

taxable_income = max(0, min(income, upper_bound) - previous_upper_bound)

Calculate the tax for the current bracket and accumulate it into 'total tax'

// Calculate the tax for the current bracket and add it to the total tax

break;

return tax / 100.0;

};

/**

TypeScript

```
class Solution:
   def calculateTax(self, brackets: List[List[int]], income: int) -> float:
       # Initialize the total tax amount to 0
        total_tax = 0
       # Initialize 'previous_upper_bound' which will hold the upper bound of the previous bracket
       previous_upper_bound = 0
       # Loop through the collected tax brackets
        for upper_bound, tax_rate in brackets:
           # Calculate the taxable income for the current bracket.
```

Update 'previous_upper_bound' to the current bracket's upper bound for the next iteration

Divide the 'total_tax' by 100 to convert the tax rate to a percentage and return the result

'max(0, min(income, upper_bound) - previous_upper_bound)' ensures that the income does not exceed the current bracket's

The provided code snippet is designed to calculate taxes based on various tax brackets and the given income. Here is the analysis of both the time complexity and space complexity of the code:

return total_tax / 100

Time and Space Complexity

contains a loop that iterates through each bracket exactly once. for upper, percent in brackets:

The time complexity of the calculateTax function is O(n), where n is the number of tax brackets. This is because the function

Inside the loop, operations are performed in constant time, including comparisons, arithmetic operations, and variable

the input.

assignments.

prev = upper

Time Complexity

Space Complexity The space complexity of the calculateTax function is 0(1). The algorithm uses a fixed amount of extra space for variables ans

```
ans = prev = 0
```

It's important to note that the input brackets, which is provided to the function, does not count towards the space complexity as it is considered input to the function and not extra space used by the function itself.

and prev. No additional space which grows with the input size is utilized, as there are no data structures dependent on the size of