# Problem 1792: Maximum Average Pass Ratio

## Problem Information

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

## Problem Description

There is a school that has classes of students and each class will be having a final exam. You are given a 2D integer array

classes

, where

classes[i] = [pass

i

, total

i

]

. You know beforehand that in the

i

th

class, there are

total

i

total students, but only

pass

i

number of students will pass the exam.

You are also given an integer

extraStudents

. There are another

extraStudents

brilliant students that are

guaranteed

to pass the exam of any class they are assigned to. You want to assign each of the

extraStudents

students to a class in a way that

maximizes

the

average

pass ratio across

all

the classes.

The

pass ratio

of a class is equal to the number of students of the class that will pass the exam divided by the total number of students of the class. The

average pass ratio

is the sum of pass ratios of all the classes divided by the number of the classes.

Return

the

maximum

possible average pass ratio after assigning the

extraStudents

students.

Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

classes = [[1,2],[3,5],[2,2]],

extraStudents

= 2

Output:

0.78333

Explanation:

You can assign the two extra students to the first class. The average pass ratio will be equal to (3/4 + 3/5 + 2/2) / 3 = 0.78333.

Example 2:

Input:

classes = [[2,4],[3,9],[4,5],[2,10]],

extraStudents

= 4

Output:

0.53485

Constraints:

1 <= classes.length <= 10

5

classes[i].length == 2

1 <= pass

i

<= total

i

<= 10

5

1 <= extraStudents <= 10

5

## Code Snippets

**C++:**

```
class Solution {
public:
double maxAverageRatio(vector<vector<int>>& classes, int extraStudents) {

}
};
```

**Java:**

```
class Solution {
public double maxAverageRatio(int[][] classes, int extraStudents) {

}
}
```

**Python3:**

```
class Solution:
def maxAverageRatio(self, classes: List[List[int]], extraStudents: int) ->
float:
```

**Python:**

```
class Solution(object):
def maxAverageRatio(self, classes, extraStudents):
"""
:type classes: List[List[int]]
:type extraStudents: int
:rtype: float
"""
```

**JavaScript:**

```
/**
* @param {number[][]} classes
* @param {number} extraStudents
* @return {number}
*/
var maxAverageRatio = function(classes, extraStudents) {

};
```

**TypeScript:**

```
function maxAverageRatio(classes: number[][], extraStudents: number): number
{

};
```

**C#:**

```
public class Solution {
public double MaxAverageRatio(int[][] classes, int extraStudents) {

}
}
```

**C:**

```
double maxAverageRatio(int** classes, int classesSize, int* classesColSize,
int extraStudents) {

}
```

**Go:**

```
func maxAverageRatio(classes [][]int, extraStudents int) float64 {

}
```

**Kotlin:**

```
class Solution {
fun maxAverageRatio(classes: Array<IntArray>, extraStudents: Int): Double {

}
}
```

**Swift:**

```
class Solution {
func maxAverageRatio(_ classes: [[Int]], _ extraStudents: Int) -> Double {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_average_ratio(classes: Vec<Vec<i32>>, extra_students: i32) -> f64
{

}
}
```

**Ruby:**

```
# @param {Integer[][]} classes
# @param {Integer} extra_students
# @return {Float}
def max_average_ratio(classes, extra_students)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $classes
* @param Integer $extraStudents
* @return Float
*/
function maxAverageRatio($classes, $extraStudents) {

}
}
```

**Dart:**

```
class Solution {
double maxAverageRatio(List<List<int>> classes, int extraStudents) {

}
}
```

**Scala:**

```
object Solution {
def maxAverageRatio(classes: Array[Array[Int]], extraStudents: Int): Double =
{

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_average_ratio(classes :: [[integer]], extra_students :: integer) ::
float
def max_average_ratio(classes, extra_students) do

end
end
```

**Erlang:**

```
-spec max_average_ratio(Classes :: [[integer()]], ExtraStudents :: integer())
-> float().
max_average_ratio(Classes, ExtraStudents) ->
.
```

### Racket:

```
(define/contract (max-average-ratio classes extraStudents)
(-> (listof (listof exact-integer?)) exact-integer? flonum?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 maxAverageRatio(vector<vector<int>>& classes, int extraStudents) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 maxAverageRatio(int[][] classes, int extraStudents) {
}
}
```

**Python3 Solution:**

```
"""
Problem: Maximum Average Pass Ratio
Difficulty: Medium
Tags: array, greedy, queue, heap

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 maxAverageRatio(self, classes: List[List[int]], extraStudents: int) ->
float:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def maxAverageRatio(self, classes, extraStudents):
"""
:type classes: List[List[int]]
:type extraStudents: int
:rtype: float
"""
```

**JavaScript Solution:**

```
/**
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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[][]} classes
* @param {number} extraStudents
* @return {number}
*/
var maxAverageRatio = function(classes, extraStudents) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 maxAverageRatio(classes: number[][], extraStudents: number): number
{

};
```

**C# Solution:**

```
/*
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 MaxAverageRatio(int[][] classes, int extraStudents) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum Average Pass Ratio
* Difficulty: Medium
* Tags: array, greedy, queue, heap
*
* 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 maxAverageRatio(int** classes, int classesSize, int* classesColSize,
int extraStudents) {

}
```

### Go Solution:

```
// Problem: Maximum Average Pass Ratio
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// 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 maxAverageRatio(classes [][]int, extraStudents int) float64 {

}
```

### Kotlin Solution:

```
class Solution {
fun maxAverageRatio(classes: Array<IntArray>, extraStudents: Int): Double {
}
}
```

### Swift Solution:

```
class Solution {
func maxAverageRatio(_ classes: [[Int]], _ extraStudents: Int) -> Double {

}
}
```

### Rust Solution:

```
// Problem: Maximum Average Pass Ratio
// Difficulty: Medium
// Tags: array, greedy, queue, heap
//
// 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 max_average_ratio(classes: Vec<Vec<i32>>, extra_students: i32) -> f64
{

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} classes
# @param {Integer} extra_students
# @return {Float}
def max_average_ratio(classes, extra_students)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $classes
* @param Integer $extraStudents
* @return Float
*/
function maxAverageRatio($classes, $extraStudents) {

}
}
```

### Dart Solution:

```
class Solution {
double maxAverageRatio(List<List<int>> classes, int extraStudents) {

}
}
```

### Scala Solution:

```
object Solution {
def maxAverageRatio(classes: Array[Array[Int]], extraStudents: Int): Double =
{

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec max_average_ratio(classes :: [[integer]], extra_students :: integer) ::
float
def max_average_ratio(classes, extra_students) do

end
end
```

### Erlang Solution:

```
-spec max_average_ratio(Classes :: [[integer()]], ExtraStudents :: integer())
-> float().
max_average_ratio(Classes, ExtraStudents) ->
.
```

### Racket Solution:

```
(define/contract (max-average-ratio classes extraStudents)
(-> (listof (listof exact-integer?)) exact-integer? flonum?)
)
```