Problem 1701: Average Waiting Time

Problem Information

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

Problem Description

There is a restaurant with a single chef. You are given an array

customers

, where

customers[i] = [arrival

i

, time

i

]:

arrival

i

is the arrival time of the

i

th

customer. The arrival times are sorted in

non-decreasing

order.

time

i

is the time needed to prepare the order of the

i

th

customer.

When a customer arrives, he gives the chef his order, and the chef starts preparing it once he is idle. The customer waits till the chef finishes preparing his order. The chef does not prepare food for more than one customer at a time. The chef prepares food for customers

in the order they were given in the input

.

Return

the

average

waiting time of all customers

. Solutions within

10

-5

from the actual answer are considered accepted.

Example 1:

Input:

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

Output:

5.00000

Explanation:

1) The first customer arrives at time 1, the chef takes his order and starts preparing it immediately at time 1, and finishes at time 3, so the waiting time of the first customer is 3 - 1 = 2. 2) The second customer arrives at time 2, the chef takes his order and starts preparing it at time 3, and finishes at time 8, so the waiting time of the second customer is 8 - 2 = 6. 3) The third customer arrives at time 4, the chef takes his order and starts preparing it at time 8, and finishes at time 11, so the waiting time of the third customer is 11 - 4 = 7. So the average waiting time = (2 + 6 + 7) / 3 = 5.

Example 2:

Input:

customers = [[5,2],[5,4],[10,3],[20,1]]

Output:

3.25000

Explanation:

1) The first customer arrives at time 5, the chef takes his order and starts preparing it immediately at time 5, and finishes at time 7, so the waiting time of the first customer is 7 - 5 = 2. 2) The second customer arrives at time 5, the chef takes his order and starts preparing it at time 7, and finishes at time 11, so the waiting time of the second customer is 11 - 5 = 6. 3) The third customer arrives at time 10, the chef takes his order and starts preparing it at time

11, and finishes at time 14, so the waiting time of the third customer is 14 - 10 = 4. 4) The fourth customer arrives at time 20, the chef takes his order and starts preparing it immediately at time 20, and finishes at time 21, so the waiting time of the fourth customer is 21 - 20 = 1. So the average waiting time = (2 + 6 + 4 + 1) / 4 = 3.25.

Constraints:

1 <= customers.length <= 10

5

1 <= arrival

i

, time

i

<= 10

4

arrival

i

<= arrival

i+1

Code Snippets

C++:

```
class Solution {
public:
double averageWaitingTime(vector<vector<int>>& customers) {
}
};
```

Java:

```
class Solution {
public double averageWaitingTime(int[][] customers) {

}
}
```

Python3:

```
class Solution:
def averageWaitingTime(self, customers: List[List[int]]) -> float:
```

Python:

```
class Solution(object):
def averageWaitingTime(self, customers):
"""
:type customers: List[List[int]]
:rtype: float
"""
```

JavaScript:

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

};
```

TypeScript:

```
function averageWaitingTime(customers: number[][]): number {

};
```

C#:

}

```
public class Solution {
public double AverageWaitingTime(int[][] customers) {

}
}
```

C:

```
double averageWaitingTime(int** customers, int customersSize, int*
customersColSize) {

}
```

Go:

```
func averageWaitingTime(customers [][]int) float64 {

}
```

Kotlin:

```
class Solution {
fun averageWaitingTime(customers: Array<IntArray>): Double {

}
}
```

Swift:

```
class Solution {
func averageWaitingTime(_ customers: [[Int]]) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn average_waiting_time(customers: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[][]} customers
# @return {Float}
def average_waiting_time(customers)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $customers
* @return Float
*/
function averageWaitingTime($customers) {

}
}
```

Dart:

```
class Solution {
double averageWaitingTime(List<List<int>> customers) {

}
}
```

Scala:

```
object Solution {
def averageWaitingTime(customers: Array[Array[Int]]): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec average_waiting_time(customers :: [[integer]]) :: float
def average_waiting_time(customers) do
```

```
end
end
```

Erlang:

```
-spec average_waiting_time(Customers :: [[integer()]]) -> float().
average_waiting_time(Customers) ->
.
```

Racket:

```
(define/contract (average-waiting-time customers)
(-> (listof (listof exact-integer?)) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Average Waiting Time
* Difficulty: Medium
* 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 averageWaitingTime(vector<vector<int>>& customers) {

}
};
```

Java Solution:

```
/**
* Problem: Average Waiting Time
* Difficulty: Medium
* 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 averageWaitingTime(int[][] customers) {

}
}
```

Python3 Solution:

```
"""
Problem: Average Waiting Time
Difficulty: Medium
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 averageWaitingTime(self, customers: List[List[int]]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def averageWaitingTime(self, customers):
"""
:type customers: List[List[int]]
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Average Waiting Time
* Difficulty: Medium
* 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[][]} customers
* @return {number}
*/
var averageWaitingTime = function(customers) {

};
```

TypeScript Solution:

```
/**
* Problem: Average Waiting Time
* Difficulty: Medium
* 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 averageWaitingTime(customers: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Average Waiting Time
* Difficulty: Medium
* 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 AverageWaitingTime(int[][] customers) {

}
}
```

C Solution:

```
/*
* Problem: Average Waiting Time
* Difficulty: Medium
* 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 averageWaitingTime(int** customers, int customersSize, int*
customersColSize) {

}
```

Go Solution:

```
// Problem: Average Waiting Time
// Difficulty: Medium
// 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 averageWaitingTime(customers [][]int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun averageWaitingTime(customers: Array<IntArray>): Double {

}
}
```

Swift Solution:

```
class Solution {
func averageWaitingTime(_ customers: [[Int]]) -> Double {

}
}
```

Rust Solution:

```
// Problem: Average Waiting Time
// Difficulty: Medium
// 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 average_waiting_time(customers: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} customers
# @return {Float}
def average_waiting_time(customers)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $customers
* @return Float
*/
function averageWaitingTime($customers) {

}
}
```

Dart Solution:

```
class Solution {
double averageWaitingTime(List<List<int>> customers) {

}
}
```

Scala Solution:

```
object Solution {
def averageWaitingTime(customers: Array[Array[Int]]): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec average_waiting_time(customers :: [[integer]]) :: float
def average_waiting_time(customers) do

end
end
```

Erlang Solution:

```
-spec average_waiting_time(Customers :: [[integer()]]) -> float().
average_waiting_time(Customers) ->
.
```

Racket Solution:

```
(define/contract (average-waiting-time customers)
(-> (listof (listof exact-integer?)) flonum?)
)
```