Problem 1491: Average Salary Excluding the Minimum and Maximum Salary

Problem Information

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

Problem Description

You are given an array of

unique

integers

salary

where

salary[i]

is the salary of the

i

th

employee.

Return

the average salary of employees excluding the minimum and maximum salary

. Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

salary = [4000,3000,1000,2000]

Output:

2500.00000

Explanation:

Minimum salary and maximum salary are 1000 and 4000 respectively. Average salary excluding minimum and maximum salary is (2000+3000) / 2 = 2500

Example 2:

Input:

salary = [1000,2000,3000]

Output:

2000.00000

Explanation:

Minimum salary and maximum salary are 1000 and 3000 respectively. Average salary excluding minimum and maximum salary is (2000) / 1 = 2000

Constraints:

3 <= salary.length <= 100

1000 <= salary[i] <= 10

6

All the integers of

salary

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
double average(vector<int>& salary) {

}
};
```

Java:

```
class Solution {
public double average(int[] salary) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function average(salary: number[]): number {

};
```

C#:

```
public class Solution {
public double Average(int[] salary) {

}
}
```

C:

```
double average(int* salary, int salarySize) {

}
```

Go:

```
func average(salary []int) float64 {

}
```

Kotlin:

```
class Solution {
fun average(salary: IntArray): Double {

}
}
```

Swift:

```
class Solution {
func average(_ salary: [Int]) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn average(salary: Vec<i32>) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[]} salary
# @return {Float}
def average(salary)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $salary
* @return Float
*/
function average($salary) {

}
}
```

Dart:

```
class Solution {
double average(List<int> salary) {

}
}
```

Scala:

```
object Solution {
def average(salary: Array[Int]): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec average(salary :: [integer]) :: float
def average(salary) do

end
end
```

Erlang:

```
-spec average(Salary :: [integer()]) -> float().
average(Salary) ->
.
```

Racket:

```
(define/contract (average salary)
(-> (listof exact-integer?) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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 average(vector<int>& salary) {

}
};
```

Java Solution:

```
/**
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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 average(int[] salary) {

}
}
```

Python3 Solution:

```
"""
Problem: Average Salary Excluding the Minimum and Maximum Salary
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 average(self, salary: List[int]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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[]} salary
* @return {number}
*/
var average = function(salary) {

};
```

TypeScript Solution:

```
/**
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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 average(salary: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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 Average(int[] salary) {

}
}
```

C Solution:

```
/*
* Problem: Average Salary Excluding the Minimum and Maximum Salary
* 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 average(int* salary, int salarySize) {

}
```

Go Solution:

```
// Problem: Average Salary Excluding the Minimum and Maximum Salary
// 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 average(salary []int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun average(salary: IntArray): Double {

}
}
```

Swift Solution:

```
class Solution {
func average(_ salary: [Int]) -> Double {

}
}
```

Rust Solution:

```
// Problem: Average Salary Excluding the Minimum and Maximum Salary
// 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 average(salary: Vec<i32>) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} salary
# @return {Float}
def average(salary)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $salary
* @return Float
*/
function average($salary) {

}
}
```

Dart Solution:

```
class Solution {
double average(List<int> salary) {

}
}
```

Scala Solution:

```
object Solution {
def average(salary: Array[Int]): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec average(salary :: [integer]) :: float
def average(salary) do

end
end
```

Erlang Solution:

```
-spec average(Salary :: [integer()]) -> float().
average(Salary) ->
.
```

Racket Solution:

```
(define/contract (average salary)
(-> (listof exact-integer?) flonum?)
)
```