Problem 2137: Pour Water Between Buckets to Make Water Levels Equal

Problem Information

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

Problem Description

You have

n

buckets each containing some gallons of water in it, represented by a

0-indexed

integer array

buckets

, where the

i

th

bucket contains

buckets[i]

gallons of water. You are also given an integer

loss

.

You want to make the amount of water in each bucket equal. You can pour any amount of water from one bucket to another bucket (not necessarily an integer). However, every time you pour

k

gallons of water, you spill

loss

percent

of

k

.

Return

the

maximum

amount of water in each bucket after making the amount of water equal.

Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

buckets = [1,2,7], loss = 80

Output:

2.00000

Explanation:

Pour 5 gallons of water from buckets[2] to buckets[0]. 5 * 80% = 4 gallons are spilled and buckets[0] only receives 5 - 4 = 1 gallon of water. All buckets have 2 gallons of water in them so return 2.

Example 2:

Input:

buckets = [2,4,6], loss = 50

Output:

3.50000

Explanation:

Pour 0.5 gallons of water from buckets[1] to buckets[0]. 0.5 * 50% = 0.25 gallons are spilled and buckets[0] only receives 0.5 - 0.25 = 0.25 gallons of water. Now, buckets = [2.25, 3.5, 6]. Pour 2.5 gallons of water from buckets[2] to buckets[0]. 2.5 * 50% = 1.25 gallons are spilled and buckets[0] only receives 2.5 - 1.25 = 1.25 gallons of water. All buckets have 3.5 gallons of water in them so return 3.5.

Example 3:

Input:

buckets = [3,3,3,3], loss = 40

Output:

3.00000

Explanation:

All buckets already have the same amount of water in them.

Constraints:

1 <= buckets.length <= 10

5

0 <= buckets[i] <= 10

5

0 <= loss <= 99

Code Snippets

C++:

```
class Solution {
public:
double equalizeWater(vector<int>& buckets, int loss) {

}
};
```

Java:

```
class Solution {
public double equalizeWater(int[] buckets, int loss) {

}
}
```

Python3:

```
class Solution:
def equalizeWater(self, buckets: List[int], loss: int) -> float:
```

Python:

```
class Solution(object):
def equalizeWater(self, buckets, loss):
"""
:type buckets: List[int]
:type loss: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number[]} buckets
* @param {number} loss
* @return {number}
*/
var equalizeWater = function(buckets, loss) {

};
```

TypeScript:

```
function equalizeWater(buckets: number[], loss: number): number {

};
```

C#:

```
public class Solution {
public double EqualizeWater(int[] buckets, int loss) {

}
}
```

C:

```
double equalizeWater(int* buckets, int bucketsSize, int loss) {

}
```

Go:

```
func equalizeWater(buckets []int, loss int) float64 {

}
```

Kotlin:

```
class Solution {
fun equalizeWater(buckets: IntArray, loss: Int): Double {

}
}
```

Swift:

```
class Solution {
func equalizeWater(_ buckets: [Int], _ loss: Int) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn equalize_water(buckets: Vec<i32>, loss: i32) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[]} buckets
# @param {Integer} loss
# @return {Float}
def equalize_water(buckets, loss)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $buckets
* @param Integer $loss
* @return Float
*/
function equalizeWater($buckets, $loss) {

}
}
```

Dart:

```
class Solution {
double equalizeWater(List<int> buckets, int loss) {

}
}
```

Scala:

```
object Solution {
def equalizeWater(buckets: Array[Int], loss: Int): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec equalize_water(buckets :: [integer], loss :: integer) :: float
def equalize_water(buckets, loss) do

end
end
```

Erlang:

```
-spec equalize_water(Buckets :: [integer()], Loss :: integer()) -> float().
equalize_water(Buckets, Loss) ->
.
```

Racket:

```
(define/contract (equalize-water buckets loss)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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 equalizeWater(vector<int>& buckets, int loss) {

}
};
```

Java Solution:

```
/**
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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 equalizeWater(int[] buckets, int loss) {

}
}
```

Python3 Solution:

```
"""
Problem: Pour Water Between Buckets to Make Water Levels Equal
Difficulty: Medium
Tags: array, search

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 equalizeWater(self, buckets: List[int], loss: int) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def equalizeWater(self, buckets, loss):
"""
:type buckets: List[int]
:type loss: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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[]} buckets
* @param {number} loss
* @return {number}
*/
var equalizeWater = function(buckets, loss) {

};
```

TypeScript Solution:

```
/**
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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 equalizeWater(buckets: number[], loss: number): number {

};
```

C# Solution:

```
/*
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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 EqualizeWater(int[] buckets, int loss) {

}
}
```

C Solution:

```
/*
* Problem: Pour Water Between Buckets to Make Water Levels Equal
* Difficulty: Medium
* Tags: array, search
*
* 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 equalizeWater(int* buckets, int bucketsSize, int loss) {

}
```

Go Solution:

```
// Problem: Pour Water Between Buckets to Make Water Levels Equal
// Difficulty: Medium
// Tags: array, search
//
// 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 equalizeWater(buckets []int, loss int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun equalizeWater(buckets: IntArray, loss: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func equalizeWater(_ buckets: [Int], _ loss: Int) -> Double {

}
}
```

Rust Solution:

```
// Problem: Pour Water Between Buckets to Make Water Levels Equal
// Difficulty: Medium
// Tags: array, search
//
// 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 equalize_water(buckets: Vec<i32>, loss: i32) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} buckets
# @param {Integer} loss
# @return {Float}
def equalize_water(buckets, loss)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $buckets
* @param Integer $loss
* @return Float
*/
function equalizeWater($buckets, $loss) {

}
}
```

Dart Solution:

```
class Solution {
double equalizeWater(List<int> buckets, int loss) {

}
}
```

Scala Solution:

```
object Solution {
def equalizeWater(buckets: Array[Int], loss: Int): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec equalize_water(buckets :: [integer], loss :: integer) :: float
def equalize_water(buckets, loss) do

end
end
```

Erlang Solution:

```
-spec equalize_water(Buckets :: [integer()], Loss :: integer()) -> float().
equalize_water(Buckets, Loss) ->
.
```

Racket Solution:

```
(define/contract (equalize-water buckets loss)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```