Problem 2028: Find Missing Observations

Problem Information

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

Problem Description

You have observations of

n + m

6-sided

dice rolls with each face numbered from

1

to

6

.

n

of the observations went missing, and you only have the observations of

m

rolls. Fortunately, you have also calculated the

average value

of the

$n + m$

rolls.

You are given an integer array

rolls

of length

m

where

rolls[i]

is the value of the

i

th

observation. You are also given the two integers

mean

and

n

.

Return

an array of length

n

containing the missing observations such that the

average value

of the

n + m

rolls is

exactly

mean

. If there are multiple valid answers, return

any of them

. If no such array exists, return

an empty array

.

The

average value

of a set of

k

numbers is the sum of the numbers divided by

k

.

Note that

mean

is an integer, so the sum of the

n + m

rolls should be divisible by

n + m

.

Example 1:

Input:

rolls = [3,2,4,3], mean = 4, n = 2

Output:

[6,6]

Explanation:

The mean of all n + m rolls is (3 + 2 + 4 + 3 + 6 + 6) / 6 = 4.

Example 2:

Input:

rolls = [1,5,6], mean = 3, n = 4

Output:

[2,3,2,2]

Explanation:

The mean of all n + m rolls is (1 + 5 + 6 + 2 + 3 + 2 + 2) / 7 = 3.

Example 3:

Input:

rolls = [1,2,3,4], mean = 6, n = 4

Output:

[]

Explanation:

It is impossible for the mean to be 6 no matter what the 4 missing rolls are.

Constraints:

m == rolls.length

1 <= n, m <= 10

5

1 <= rolls[i], mean <= 6

Code Snippets

C++:

```
class Solution {
public:
vector<int> missingRolls(vector<int>& rolls, int mean, int n) {

}
};
```

Java:

```
class Solution {
public int[] missingRolls(int[] rolls, int mean, int n) {

}
}
```

Python3:

```
class Solution:
def missingRolls(self, rolls: List[int], mean: int, n: int) -> List[int]:
```

Python:

```
class Solution(object):
def missingRolls(self, rolls, mean, n):
"""
:type rolls: List[int]
:type mean: int
:type n: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} rolls
* @param {number} mean
* @param {number} n
* @return {number[]}
*/
var missingRolls = function(rolls, mean, n) {

};
```

TypeScript:

```
function missingRolls(rolls: number[], mean: number, n: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MissingRolls(int[] rolls, int mean, int n) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* missingRolls(int* rolls, int rollsSize, int mean, int n, int*
returnSize) {

}
```

Go:

```
func missingRolls(rolls []int, mean int, n int) []int {

}
```

Kotlin:

```
class Solution {
fun missingRolls(rolls: IntArray, mean: Int, n: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func missingRolls(_ rolls: [Int], _ mean: Int, _ n: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn missing_rolls(rolls: Vec<i32>, mean: i32, n: i32) -> Vec<i32> {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} rolls
# @param {Integer} mean
# @param {Integer} n
# @return {Integer[]}
def missing_rolls(rolls, mean, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $rolls
* @param Integer $mean
* @param Integer $n
* @return Integer[]
*/
function missingRolls($rolls, $mean, $n) {

}
}
```

Dart:

```
class Solution {
List<int> missingRolls(List<int> rolls, int mean, int n) {

}
}
```

Scala:

```
object Solution {
def missingRolls(rolls: Array[Int], mean: Int, n: Int): Array[Int] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec missing_rolls(rolls :: [integer], mean :: integer, n :: integer) ::
[integer]
def missing_rolls(rolls, mean, n) do

end
end
```

Erlang:

```
-spec missing_rolls(Rolls :: [integer()], Mean :: integer(), N :: integer())
-> [integer()].
missing_rolls(Rolls, Mean, N) ->
.
```

Racket:

```
(define/contract (missing-rolls rolls mean n)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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:
vector<int> missingRolls(vector<int>& rolls, int mean, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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 int[] missingRolls(int[] rolls, int mean, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Missing Observations
Difficulty: Medium
Tags: array, math

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 missingRolls(self, rolls: List[int], mean: int, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def missingRolls(self, rolls, mean, n):
"""
:type rolls: List[int]
:type mean: int
:type n: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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[]} rolls
* @param {number} mean
* @param {number} n
* @return {number[]}
*/
var missingRolls = function(rolls, mean, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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 missingRolls(rolls: number[], mean: number, n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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 int[] MissingRolls(int[] rolls, int mean, int n) {

}
}
```

C Solution:

```
/*
* Problem: Find Missing Observations
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* missingRolls(int* rolls, int rollsSize, int mean, int n, int*
```

```
returnSize) {

}
```

Go Solution:

```
// Problem: Find Missing Observations
// Difficulty: Medium
// Tags: array, math
//
// 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 missingRolls(rolls []int, mean int, n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun missingRolls(rolls: IntArray, mean: Int, n: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func missingRolls(_ rolls: [Int], _ mean: Int, _ n: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Missing Observations
// Difficulty: Medium
// Tags: array, math
//
// 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 missing_rolls(rolls: Vec<i32>, mean: i32, n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} rolls
# @param {Integer} mean
# @param {Integer} n
# @return {Integer[]}
def missing_rolls(rolls, mean, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $rolls
* @param Integer $mean
* @param Integer $n
* @return Integer[]
*/
function missingRolls($rolls, $mean, $n) {

}
}
```

Dart Solution:

```
class Solution {
List<int> missingRolls(List<int> rolls, int mean, int n) {

}
}
```

Scala Solution:

```
object Solution {
def missingRolls(rolls: Array[Int], mean: Int, n: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec missing_rolls(rolls :: [integer], mean :: integer, n :: integer) ::
[integer]
def missing_rolls(rolls, mean, n) do

end
end
```

Erlang Solution:

```
-spec missing_rolls(Rolls :: [integer()], Mean :: integer(), N :: integer())
-> [integer()].
missing_rolls(Rolls, Mean, N) ->
.
```

Racket Solution:

```
(define/contract (missing-rolls rolls mean n)
(-> (listof exact-integer?) exact-integer? exact-integer? (listof
exact-integer?))
)
```