Problem 1093: Statistics from a Large Sample

Problem Information

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

Problem Description

You are given a large sample of integers in the range

[0, 255]

. Since the sample is so large, it is represented by an array

count

where

count[k]

is the

number of times

that

k

appears in the sample.

Calculate the following statistics:

minimum

: The minimum element in the sample.

maximum

: The maximum element in the sample.

mean

: The average of the sample, calculated as the total sum of all elements divided by the total number of elements.

median

:

If the sample has an odd number of elements, then the

median

is the middle element once the sample is sorted.

If the sample has an even number of elements, then the

median

is the average of the two middle elements once the sample is sorted.

mode

: The number that appears the most in the sample. It is guaranteed to be

unique

.

Return

the statistics of the sample as an array of floating-point numbers

[minimum, maximum, mean, median, mode]

. Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input:

count = [0,1,3,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0]

Output:

[1.00000,3.00000,2.37500,2.50000,3.00000]

Explanation:

The sample represented by count is [1,2,2,2,3,3,3,3]. The minimum and maximum are 1 and 3 respectively. The mean is (1+2+2+2+3+3+3+3) / 8 = 19 / 8 = 2.375. Since the size of the sample is even, the median is the average of the two middle elements 2 and 3, which is 2.5. The mode is 3 as it appears the most in the sample.

Example 2:

Input:

count = [0,4,3,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0]

Output:

[1.00000,4.00000,2.18182,2.00000,1.00000]

Explanation:

The sample represented by count is [1,1,1,1,2,2,2,3,3,4,4]. The minimum and maximum are 1 and 4 respectively. The mean is (1+1+1+1+2+2+2+3+3+4+4) / 11 = 24 / 11 = 2.18181818... (for display purposes, the output shows the rounded number 2.18182). Since the size of the sample is odd, the median is the middle element 2. The mode is 1 as it appears the most in the sample.

Constraints:

count.length == 256

0 <= count[i] <= 10

9

1 <= sum(count) <= 10

9

The mode of the sample that

count

represents is

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<double> sampleStats(vector<int>& count) {

}
};
```

Java:

```
class Solution {
public double[] sampleStats(int[] count) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sampleStats(count: number[]): number[] {

};
```

C#:

```
public class Solution {
public double[] SampleStats(int[] count) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
double* sampleStats(int* count, int countSize, int* returnSize) {

}
```

Go:

```
func sampleStats(count []int) []float64 {

}
```

Kotlin:

```
class Solution {
fun sampleStats(count: IntArray): DoubleArray {

}
}
```

Swift:

```
class Solution {
func sampleStats(_ count: [Int]) -> [Double] {

}
}
```

Rust:

```
impl Solution {
pub fn sample_stats(count: Vec<i32>) -> Vec<f64> {

}
}
```

Ruby:

```
# @param {Integer[]} count
# @return {Float[]}
def sample_stats(count)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $count
* @return Float[]
*/
function sampleStats($count) {

}
}
```

Dart:

```
class Solution {
List<double> sampleStats(List<int> count) {

}
}
```

Scala:

```
object Solution {
def sampleStats(count: Array[Int]): Array[Double] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sample_stats(count :: [integer]) :: [float]
def sample_stats(count) do

end
end
```

Erlang:

```
-spec sample_stats(Count :: [integer()]) -> [float()].
sample_stats(Count) ->
.
```

Racket:

```
(define/contract (sample-stats count)
(-> (listof exact-integer?) (listof flonum?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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:
vector<double> sampleStats(vector<int>& count) {
}
};
```

Java Solution:

```
/**
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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[] sampleStats(int[] count) {

}
}
```

Python3 Solution:

```
"""
Problem: Statistics from a Large Sample
Difficulty: Medium
Tags: array, math, 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 sampleStats(self, count: List[int]) -> List[float]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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[]} count
* @return {number[]}
*/
var sampleStats = function(count) {

};
```

TypeScript Solution:

```
/**
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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 sampleStats(count: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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[] SampleStats(int[] count) {

}
}
```

C Solution:

```
/*
* Problem: Statistics from a Large Sample
* Difficulty: Medium
* Tags: array, math, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
double* sampleStats(int* count, int countSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Statistics from a Large Sample
// Difficulty: Medium
// Tags: array, math, 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 sampleStats(count []int) []float64 {

}
```

Kotlin Solution:

```
class Solution {
fun sampleStats(count: IntArray): DoubleArray {

}
}
```

Swift Solution:

```
class Solution {
func sampleStats(_ count: [Int]) -> [Double] {

}
}
```

Rust Solution:

```
// Problem: Statistics from a Large Sample
// Difficulty: Medium
// Tags: array, math, 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 sample_stats(count: Vec<i32>) -> Vec<f64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} count
# @return {Float[]}
def sample_stats(count)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $count
* @return Float[]
*/
function sampleStats($count) {

}
}
```

Dart Solution:

```
class Solution {
List<double> sampleStats(List<int> count) {

}
}
```

Scala Solution:

```
object Solution {
def sampleStats(count: Array[Int]): Array[Double] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sample_stats(count :: [integer]) :: [float]
def sample_stats(count) do
```

```
end
end
```

Erlang Solution:

```
-spec sample_stats(Count :: [integer()]) -> [float()].
sample_stats(Count) ->
.
```

Racket Solution:

```
(define/contract (sample-stats count)
(-> (listof exact-integer?) (listof flonum?))
)
```