Problem 1467: Probability of a Two Boxes Having The Same Number of Distinct Balls

Problem Information

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

Problem Description

Given

2n

balls of

k

distinct colors. You will be given an integer array

balls

of size

k

where

balls[i]

is the number of balls of color

i

.

All the balls will be

shuffled uniformly at random

, then we will distribute the first

n

balls to the first box and the remaining

n

balls to the other box (Please read the explanation of the second example carefully).

Please note that the two boxes are considered different. For example, if we have two balls of colors

a

and

b

, and two boxes

[]

and

()

, then the distribution

[a] (b)

is considered different than the distribution

[b] (a)

(Please read the explanation of the first example carefully).

Return

the probability

that the two boxes have the same number of distinct balls. Answers within

10

-5

of the actual value will be accepted as correct.

Example 1:

Input:

balls = [1,1]

Output:

1.00000

Explanation:

Only 2 ways to divide the balls equally: - A ball of color 1 to box 1 and a ball of color 2 to box 2 - A ball of color 2 to box 1 and a ball of color 1 to box 2 In both ways, the number of distinct colors in each box is equal. The probability is 2/2 = 1

Example 2:

Input:

balls = [2,1,1]

Output:

0.66667

Explanation:

We have the set of balls [1, 1, 2, 3] This set of balls will be shuffled randomly and we may have one of the 12 distinct shuffles with equal probability (i.e. 1/12): [1,1 / 2,3], [1,1 / 3,2], [1,2 / 1,3], [1,2 / 3,1], [1,3 / 1,2], [1,3 / 2,1], [2,1 / 1,3], [2,1 / 3,1], [2,3 / 1,1], [3,1 / 1,2], [3,1 / 2,1], [3,2 / 1,1] After that, we add the first two balls to the first box and the second two balls to the second box. We can see that 8 of these 12 possible random distributions have the same number of distinct colors of balls in each box. Probability is 8/12 = 0.66667

Example 3:

Input:

balls = [1,2,1,2]

Output:

0.60000

Explanation:

The set of balls is [1, 2, 2, 3, 4, 4]. It is hard to display all the 180 possible random shuffles of this set but it is easy to check that 108 of them will have the same number of distinct colors in each box. Probability = 108 / 180 = 0.6

Constraints:

1 <= balls.length <= 8

1 <= balls[i] <= 6

sum(balls)

is even.

Code Snippets

C++:

```
class Solution {
public:
double getProbability(vector<int>& balls) {

}
};
```

Java:

```
class Solution {
public double getProbability(int[] balls) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getProbability(balls: number[]): number {

};
```

C#:

```
public class Solution {
public double GetProbability(int[] balls) {

}
}
```

C:

```
double getProbability(int* balls, int ballsSize) {

}
```

Go:

```
func getProbability(balls []int) float64 {

}
```

Kotlin:

```
class Solution {
fun getProbability(balls: IntArray): Double {

}
}
```

Swift:

```
class Solution {
func getProbability(_ balls: [Int]) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn get_probability(balls: Vec<i32>) -> f64 {
}
}
```

Ruby:

```
# @param {Integer[]} balls
# @return {Float}
def get_probability(balls)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $balls
* @return Float
*/
function getProbability($balls) {

}
}
```

Dart:

```
class Solution {
double getProbability(List<int> balls) {

}
}
```

Scala:

```
object Solution {
def getProbability(balls: Array[Int]): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_probability(balls :: [integer]) :: float
def get_probability(balls) do

end
end
```

Erlang:

```
-spec get_probability(Balls :: [integer()]) -> float().
get_probability(Balls) ->
.
```

Racket:

```
(define/contract (get-probability balls)
(-> (listof exact-integer?) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Probability of a Two Boxes Having The Same Number of Distinct
Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
double getProbability(vector<int>& balls) {

}
};
```

Java Solution:

```
/**
* Problem: Probability of a Two Boxes Having The Same Number of Distinct
Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public double getProbability(int[] balls) {

}
}
```

Python3 Solution:

```
"""
Problem: Probability of a Two Boxes Having The Same Number of Distinct Balls
Difficulty: Hard
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def getProbability(self, balls: List[int]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Probability of a Two Boxes Having The Same Number of Distinct Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} balls
* @return {number}
*/
var getProbability = function(balls) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Probability of a Two Boxes Having The Same Number of Distinct Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function getProbability(balls: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Probability of a Two Boxes Having The Same Number of Distinct
Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public double GetProbability(int[] balls) {

}
}
```

C Solution:

```
/*
* Problem: Probability of a Two Boxes Having The Same Number of Distinct
Balls
* Difficulty: Hard
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

double getProbability(int* balls, int ballsSize) {

}
```

Go Solution:

```
// Problem: Probability of a Two Boxes Having The Same Number of Distinct
Balls
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func getProbability(balls []int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun getProbability(balls: IntArray): Double {

}
}
```

Swift Solution:

```
class Solution {
func getProbability(_ balls: [Int]) -> Double {

}
}
```

Rust Solution:

```
// Problem: Probability of a Two Boxes Having The Same Number of Distinct Balls
// Difficulty: Hard
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn get_probability(balls: Vec<i32>) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} balls
# @return {Float}
def get_probability(balls)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $balls
* @return Float
*/
function getProbability($balls) {

}
}
```

Dart Solution:

```
class Solution {
double getProbability(List<int> balls) {

}
}
```

Scala Solution:

```
object Solution {
def getProbability(balls: Array[Int]): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_probability(balls :: [integer]) :: float
def get_probability(balls) do

end
```

```
end
```

Erlang Solution:

```
-spec get_probability(Balls :: [integer()]) -> float().
get_probability(Balls) ->
.
```

Racket Solution:

```
(define/contract (get-probability balls)
(-> (listof exact-integer?) flonum?)
)
```