Problem 2465: Number of Distinct Averages

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of

even

length.

As long as

nums

is

not

empty, you must repetitively:

Find the minimum number in

nums

and remove it.

Find the maximum number in

nums

and remove it.

Calculate the average of the two removed numbers.

The

average

of two numbers

a

and

b

is

(a + b) / 2

.

For example, the average of

2

and

3

is

(2 + 3) / 2 = 2.5

.

Return

the number of

distinct

averages calculated using the above process

.

Note

that when there is a tie for a minimum or maximum number, any can be removed.

Example 1:

Input:

nums = [4,1,4,0,3,5]

Output:

2

Explanation:

1. Remove 0 and 5, and the average is (0 + 5) / 2 = 2.5. Now, nums = [4,1,4,3]. 2. Remove 1 and 4. The average is (1 + 4) / 2 = 2.5, and nums = [4,3]. 3. Remove 3 and 4, and the average is (3 + 4) / 2 = 3.5. Since there are 2 distinct numbers among 2.5, 2.5, and 3.5, we return 2.

Example 2:

Input:

nums = [1,100]

Output:

1

Explanation:

There is only one average to be calculated after removing 1 and 100, so we return 1.

Constraints:

2 <= nums.length <= 100

nums.length

is even.

0 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int distinctAverages(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int distinctAverages(int[] nums) {

}
}
```

Python3:

```
class Solution:
def distinctAverages(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def distinctAverages(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function distinctAverages(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int DistinctAverages(int[] nums) {

}
}
```

C:

```
int distinctAverages(int* nums, int numsSize) {

}
```

Go:

```
func distinctAverages(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun distinctAverages(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func distinctAverages(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn distinct_averages(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def distinct_averages(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function distinctAverages($nums) {

}
}
```

Dart:

```
class Solution {
int distinctAverages(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def distinctAverages(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec distinct_averages(nums :: [integer]) :: integer
def distinct_averages(nums) do

end
end
```

Erlang:

```
-spec distinct_averages(Nums :: [integer()]) -> integer().
distinct_averages(Nums) ->
.
```

Racket:

```
(define/contract (distinct-averages nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int distinctAverages(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int distinctAverages(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Distinct Averages
Difficulty: Easy
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def distinctAverages(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @return {number}
*/
var distinctAverages = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function distinctAverages(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int DistinctAverages(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Distinct Averages
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int distinctAverages(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Number of Distinct Averages
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func distinctAverages(nums []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun distinctAverages(nums: IntArray): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func distinctAverages(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Distinct Averages
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn distinct_averages(nums: Vec<i32>) -> i32 {


}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def distinct_averages(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function distinctAverages($nums) {

}
}
```

Dart Solution:

```
class Solution {
int distinctAverages(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def distinctAverages(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec distinct_averages(nums :: [integer]) :: integer
def distinct_averages(nums) do

end
end
```

Erlang Solution:

```
-spec distinct_averages(Nums :: [integer()]) -> integer().
distinct_averages(Nums) ->
.
```

Racket Solution:

```
(define/contract (distinct-averages nums)
(-> (listof exact-integer?) exact-integer?)
)
```