Problem 3194: Minimum Average of Smallest and Largest Elements

Problem Information

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

Problem Description

You have an array of floating point numbers

averages

which is initially empty. You are given an array

nums

of

n

integers where

n

is even.

You repeat the following procedure

n / 2

times:

Remove the

smallest

element,

minElement

, and the

largest

element

maxElement

, from

nums

.

Add

(minElement + maxElement) / 2

to

averages

.

Return the

minimum

element in

averages

.

Example 1:

Input:

nums = [7,8,3,4,15,13,4,1]

Output:

5.5

Explanation:

step

nums

averages

0

[7,8,3,4,15,13,4,1]

[]

1

[7,8,3,4,13,4]

[8]

2

[7,8,4,4]

[8,8]

3

[7,4]

[8,8,6]

4

[]

[8,8,6,5.5]

The smallest element of averages, 5.5, is returned.

Example 2:

Input:

nums = [1,9,8,3,10,5]

Output:

5.5

Explanation:

step

nums

averages

0

[1,9,8,3,10,5]

[]

1

[9,8,3,5]

[5.5]

2

[8,5]

[5.5,6]

3

[]

[5.5,6,6.5]

Example 3:

Input:

nums = [1,2,3,7,8,9]

Output:

5.0

Explanation:

step

nums

averages

0

[1,2,3,7,8,9]

[]

1

[2,3,7,8]

[5]

2

[3,7]

[5,5]

3

[]

[5,5,5]

Constraints:

2 <= n == nums.length <= 50

n

is even.

1 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public double minimumAverage(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public double MinimumAverage(int[] nums) {
}
}
```

C:

```
double minimumAverage(int* nums, int numsSize) {

}
```

Go:

```
func minimumAverage(nums []int) float64 {

}
```

Kotlin:

```
class Solution {
fun minimumAverage(nums: IntArray): Double {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn minimum_average(nums: Vec<i32>) -> f64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
double minimumAverage(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-average nums)
(-> (listof exact-integer?) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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 minimumAverage(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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 minimumAverage(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Average of Smallest and Largest Elements
Difficulty: Easy
Tags: array, 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 minimumAverage(self, nums: List[int]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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[]} nums
* @return {number}
*/
var minimumAverage = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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 minimumAverage(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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 MinimumAverage(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Average of Smallest and Largest Elements
* Difficulty: Easy
* Tags: array, 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
*/

double minimumAverage(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Average of Smallest and Largest Elements
// Difficulty: Easy
// Tags: array, 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 minimumAverage(nums []int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumAverage(nums: IntArray): Double {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Average of Smallest and Largest Elements
// Difficulty: Easy
// Tags: array, 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 minimum_average(nums: Vec<i32>) -> f64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Float
*/
function minimumAverage($nums) {

}
}
```

Dart Solution:

```
class Solution {
double minimumAverage(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-average nums)
(-> (listof exact-integer?) flonum?)
)
```